Push Job Specification

Chris Brown <cb@opscode.com>
Kevin Smith <kevin@opscode.com>

March 28, 2012

1 Overview

This specification describes the ability to execute on-demand chef-client via knife. For brevity's sake this feature will be referred to as "push job" or "push jobs".

The concept of push jobs is quite simple. A user is able to select some subset of nodes managed by their Private Chef organization, specify an action or command for those nodes to execute, and track the status of each node as it executes the request.

A bit of complexity and bookkeeping lurks underneath push job's simplicity. Managed nodes will need to a way to reliably monitor the job coordinators so they can cope with outages. Job coordinators also need to monitor the status of each managed node so commands are only sent to available nodes and to track job progress.

The remainder of this document will attempt to describe this feature in enough detail to allow a reasonable and scalable implementation.

2 Assumptions

2.1 Connectivity

1. Managed nodes **MUST** be reachable via a TCP-enabled network interface.
2. Managed nodes **MUST** be able to accept incoming TCP connections.
3. Managed nodes **MUST** be able to connect to the heartbeat and job coordination components inside Chef server.

2.2 Data format & Storage

1. All messages will be formatted as legal JSON.
2. The database is the canonical store of all application data.

2.3 Scalability & Security

1. Push jobs will be deployed in Private Chef only.
2. Push jobs will not be deployed in Hosted Chef.
3. The design must scale up to 8,000 managed nodes.
4. Push jobs will honor the same security guarantees made by the Chef REST API.

3 Architecture

3.1 Communications

Managed nodes and server components will communicate using ZeroMQ messaging. Communication can be separated into two categories: heartbeat and job execution. The heartbeat channel is used by the Chef server to detect when managed nodes are offline and, therefore, unavailable for job execution. Managed nodes also use the heartbeat channel to detect when the server is unavailable.

The job execution channel is used by the Chef server to send job execution requests to managed nodes. Managed nodes used the execution channel to send job-related messages such as acknowledging jobs, sending progress updates, and reporting final results.

3.1.1 Heartbeat Channel

PUB and SUB sockets are used because they automatically and scalably manage the fanin the server requires to monitor nodes as well as the fanout the server needs to broadcast its heartbeat to all the nodes.

The details of how nodes and servers discover and connect to each other's PUB and SUB sockets is covered in Server and Client Discovery.

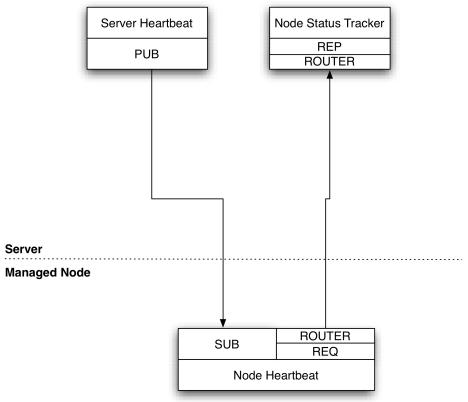

Figure 1: ZeroMQ sockets

3.1.2 Command Channel

TBD

3.2 Message Formats

3.2.1 Why JSON?

All push job messages are valid JSON hashes. Each message contains, as the first element in the hash an entry with the string key "type" and a string value denoting the message's type. Valid types are: heartbeat and TBD.

Push jobs are able to use JSON because ZeroMQ handles packet fragmentation and reassembly. Applications built using ZeroMQ are guaranteed to receive entire message payloads. This frees application developers from worrying about low level details and permits focus on formats and protocols.

Using JSON as the message format also improves the debuggability and loggability of the system. A binary protocol, such as Protocol Buffers or msgpack, would be more efficient but would also substantially increase the effort required to debug and support the system.

3.2.2 Heartbeat

```
{
  "type":  "heartbeat",
  "host":  "node123.foo.com",
  "sig":   "yYy96kyXcpV840fCW7O2Nw=="
}
```

type denotes the message type

host the sender's hostname

sig Base64 encoded cryptographic signature of the stringified JSON hash containing only the above fields.[1]

[1] Public key signatures are used to verify the sender's identity and provide some amount of message tamper detection.

3.3 Protocols

3.3.1 Heartbeat

Liveness detection in a distributed system is a notoriously difficult problem. The most common approach is to arrange for two parties to exchange heartbeat messages on a regular interval. Let's call these two parties 'A' and 'B'. Both A and B are considered 'online' while they are able to exchange heartbeat messages. If A fails to receive heartbeats from B for some number of consecutive intervals then A will consider B 'offline' and not route any traffic to B. A will update B's status to 'online' once A starts receiving heartbeats from B again.

This is the scheme used by this design. The Private Chef server sends out regular heartbeats to managed nodes via ZeroMQ PubSub. Managed nodes send their heartbeats over the command channel using the node status tracker's ZeroMQ identity. The identity is required so the message is correctly routed. See the Heartbeat Channel section for a visual representation of the message flows and ZeroMQ sockets.

3.4 Server and Client Discovery

3.4.1 REST endpoint (perhaps on /nodes?) to supply all config data in JSON format

```
{
  "type": "config",
  "host": "opc1.opscode.com",
  "push_jobs": {
                 "heartbeat": "10.10.1.5:10000",
                 "command": "10.10.1.5:10001",
                 "heartbeat_interval": 15,
                 "offline_threshold": 3,
                 "online_threshold": 2
               },
  "public_key": "AAAAB3NzaC1kc3MAAACBAIZbwlySffbB
                 5msSUH8JzLLXo/v03JBCWr13fVTjWYpc
                 cdbi/xL3IK/Jw8Rm3bGhnpwCAqBtsLvZ
                 OcqXrc2XuKBYjiKWzigBMC7wC9dUDGwDl
                 2aZ89B0jn2QPRWZuCAkxm6sKpefu++VPR
                 RZF+iyZqFwS0wVKtl97T0gwWlzAJYpAAA
                 AFQDIipDNo83e8RRp7Fits0DSy0DCpwAA
```

```
                AIB01BwXg9WSfU0mwzz/0+5Gb/TMAxfkD
                yucbcpJNncpRtr9Jb+9GjeZIbqkBQAqwg
                dbEjviRbUAuSawNSCdtnMgWD2NXkBKEde",
  "sig":   "yYy96kyXcpV840fCW7O2Nw=="

}
```

type message type

host sender's host name (Private Chef server)

push_jobs/heartbeat IP address and port of the server's ZeroMQ heartbeat channel

push_jobs/command IP address and port of the server's ZeroMQ command channel

push_jobs/heartbeat_interval Interval, in seconds, between heartbeat messages

push_jobs/offline_threshold How many heartbeat intervals must be missed before the other end is considered offline

sig Base64 encoded cryptographic signature of the stringified JSON hash containing only the above fields[1]